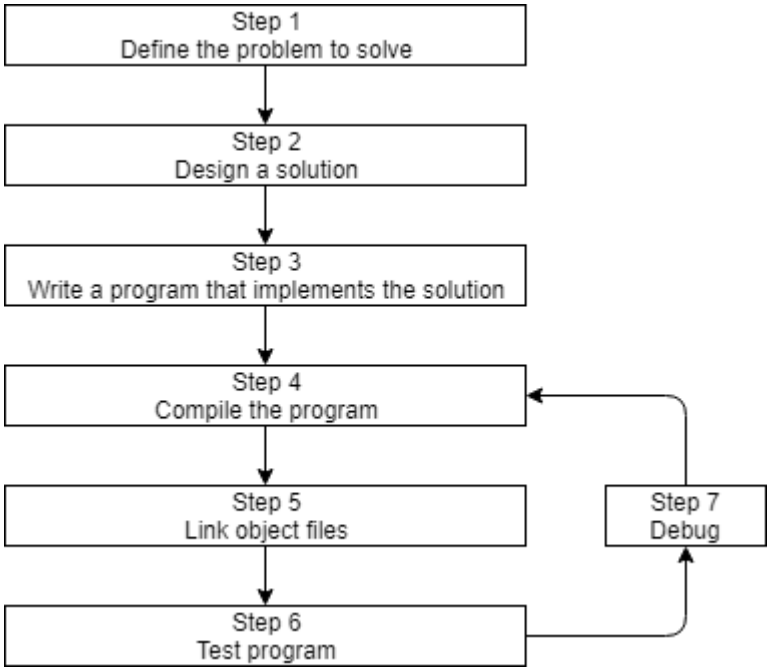
0.5 — Introduction to the compiler, linker, and libraries

ALEX[1] FEBRUARY 17, 2024

Continuing our discussion of this diagram from the previous lesson (0.4 -- Introduction to C++ development (https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/)[2]):



Let's discuss steps 4-7.

Step 4: Compiling your source code

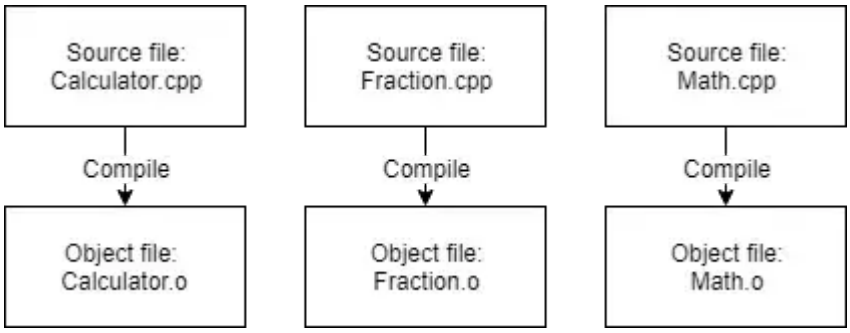
In order to compile C++ source code files, we use a C++ compiler. The C++ compiler sequentially goes through each source code (.cpp) file in your program and does two important tasks:

First, the compiler checks your C++ code to make sure it follows the rules of the C++ language. If it does not, the compiler will give you an error (and the corresponding line number) to help pinpoint what needs fixing. The compilation process will also be aborted until the error is fixed.

Second, the compiler translates your C++ code into machine language instructions. These instructions are stored in an intermediate file called an **object file**. The object file also contains metadata that is required or useful in subsequent steps.

Object files are typically named *name.o* or *name.obj*, where *name* is the same name as the .cpp file it was produced from.

For example, if your program had 3 .cpp files, the compiler would generate 3 object files:



C++ compilers are available for many different operating systems. We will discuss installing a compiler shortly, so there is no need to do so now.

Step 5: Linking object files and libraries

After the compiler has successfully finished, another program called the **linker** kicks in. The linker's job is to combine all of the object files and produce the desired output file (typically an executable file). This process is called **linking**.

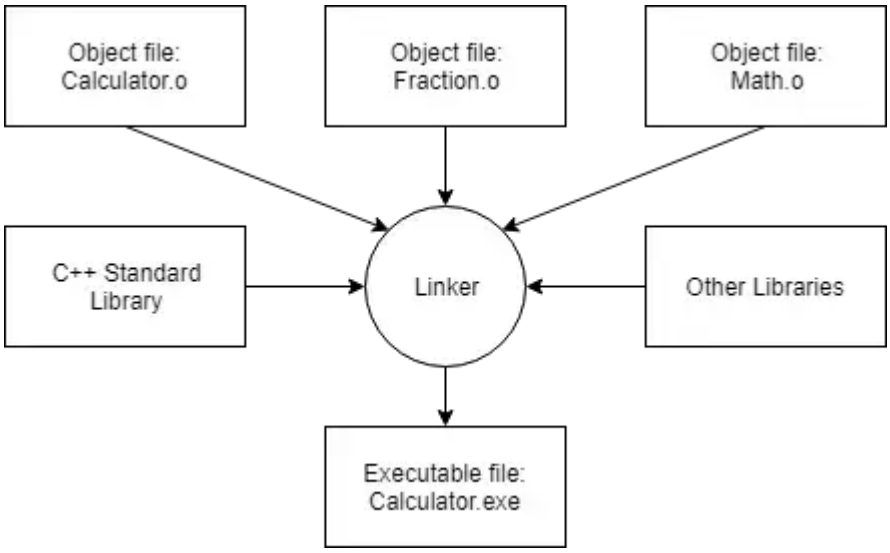
First, the linker reads in each of the object files generated by the compiler and makes sure they are valid.

Second, the linker ensures all cross-file dependencies are resolved properly. For example, if you define something in one .cpp file, and then use it in a different .cpp file, the linker connects the two together. If the linker is unable to connect a reference to something with its definition, you'll get a linker error, and the linking process will abort.

Third, the linker also is capable of linking library files. A **library file** is a collection of precompiled code that has been "packaged up" for reuse in other programs.

C++ comes with an extensive library called the **C++ Standard Library** (usually shortened to *standard library*) that provides a set of useful capabilities for use in your programs. One of the most commonly used parts of the C++ standard library is the *iostream library*, which contains functionality for printing text on a monitor and getting keyboard input from a user. Almost every C++ program written utilizes the standard library in some form, so it's very common for the standard library to get linked into your programs. Most linkers will automatically link in the standard library as soon as you use any part of it, so this generally isn't something you need to worry about.

You can also optionally link other libraries. For example, if you were going to write a program that played sounds, you probably would not want to write your own code to read in the sound files from disk, check to ensure they were valid, or figure out how to route the sound data to the operating system or hardware to play through the speaker -- that would be a lot of work! Instead, you'd probably download a library that already knew how to do those things, and use that. We'll talk about how to link in libraries (and create your own!) in the appendix.



Once the linker has finished linking all the object files and libraries, then (assuming all goes well) you will have an executable file that you can run.

Building

Because there are multiple steps involved, the term **building** is often used to refer to the full process of converting source code files into an executable that can be run. A specific executable produced as the result of building is sometimes called a **build**.

For advanced readers

For complex projects, build automation tools (such as **make** or **build2**) are often used to help automate the process of building programs and running automated tests. While such tools are powerful and flexible, because they are not part of the C++ core language, nor do you need to use them to proceed, we'll not discuss them as part of this tutorial series.

Steps 6 & 7: Testing and Debugging

This is the fun part (hopefully)! You are able to run your executable and see whether it produces the output you were expecting!

If your program runs but doesn't work correctly, then it's time for some debugging to figure out what's wrong. We will discuss how to test your programs and how to debug them in more detail soon.

Integrated development environments (IDEs)

Note that steps 3, 4, 5, and 7 all involve software programs that must be installed (editor, compiler, linker, debugger). While you can use separate programs for each of these activities, a software package known as an integrated development environment (IDE) bundles and integrates all of these features together. We'll discuss IDEs, and install one, in the next section.

Next lesson

0.6 Installing an Integrated Development Environment (IDE)

4

Back to table of contents

5

Previous lesson

0.4 Introduction to C++ development

2

6

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[9] are connected to your provided email address.

176 COMMENTS Most Voted

Muhammad Essameldin

March 26, 2024 6:06 am

well done

0 Reply

habibi lover
March 25, 2024 5:55 pm

Love this fr

0 Reply

Butcher
March 25, 2024 1:00 am

thanks for this amazing free tutorial, i have zero knownledge about programming and i hope so i can learn it with this site

0 Reply

Simon
March 19, 2024 12:12 pm

I'm learning more here than in my university I pay almost $900 a term for four terms, so around $3,500 a year. Also the textbooks used and provided to us, well I paid for them, aren't even that student-friendly. Meaning it is too complicated, so much information that you couldn't understand. I'm glad I discovered this website, in the next few months, I'm sure to improve.

0 Reply

SID
March 14, 2024 11:03 am

Ok many C++ files are connected through linker for creating and executable so what is the flow by which it decides which C++ file goes first, and which goes last and whether we create the linker program ourself or it is some other third party system or it is done by IDE after compiling and also do we need to also specify the linker which files goes first or last so they get executed properly

0 Reply

Zuyomi
Reply to SID [10] March 17, 2024 4:59 pm

If I'm not wrong, In C++, when linking multiple files to create an executable, the order in which they are linked generally doesn't matter unless there are dependencies between them. This process is usually managed automatically by build systems like CMake or by IDEs such as Visual Studio or Xcode. These tools analyze dependencies and ensure that

everything is linked in the correct order. However, if you encounter specific scenarios where manual control over the linking order is necessary, you can specify it yourself during the linking phase. But in most cases, manual intervention is not needed as modern build systems and IDEs handle this seamlessly.

0 Reply

NotLod
March 7, 2024 6:02 pm

I love this content, i learning more than the schools or other things related.

Thanks for create this web page.

0 Reply

Michelle
March 4, 2024 10:00 am

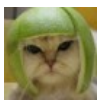
Just want to say I love your way of explaining. Thank you so much for this site! I was afraid of learning C++ because it seemed very daunting, but your explanation of confusing concepts is easy to understand.

2 Reply

rufus
February 16, 2024 10:47 am

I'm a little confused, in lesson 0.2 compilers are said to produce a stand alone executable program. Are these "object files" the programs?

0 Reply



Alex Author
Reply to rufus [11] February 17, 2024 6:03 pm

No. I tweaked the language in 0.2 and here to help clarify. For each C++ source file, the C++ compiler translates the C++ code in that file into machine language and stores it in an object file. The linker then combines all the object files into an executable file that we can run.

We can say that the object files contain part (or all) of the machine language instructions for the program, but they are not in an executable format, so they cannot be run directly.

7 Reply

S.E.

January 23, 2024 5:54 am

You say "you probably would not want to write your own code to read in the sound files from disk, check to ensure they were valid, or figure out how to route the sound data to the operating system or hardware to play through the speaker -- that would be a lot of work!" and while, it very nice that these kinds of libraries exist, I do have a strong interest in learning the processes that are involved with interfacing with hardware, as I want to be able to program my own peripherals or other electronic projects. It's funny specifically that you mention audio though, as the main program that I have always wanted to make is an audio mixer that will allow the user to define which programs are playing from which audio devices by way of a input/output digital cable based GUI. Though I do understand I am a long way from reaching that goal, it is something I have thought about for years and hope someday to achieve.

Either way, I am excited to proceed. I will need to start somewhere, so I am starting here!

8 Reply

SID

Reply to S.E. [12] March 14, 2024 10:54 am

Go on mate I also joined with a dream of making my favorite strategy game in future

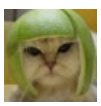
0 Reply

IceFloe

January 21, 2024 1:51 pm

I understood correctly that when assembling, there should be main. cpp, which starts the chain of other cpp files. And when building and turning the source code into an executable file obj, is this connection preserved? Can there be multiple main. cpp? Also, how can I view the source code of an already compiled file, or should I save the source code in advance?

1 Reply



Alex Author

Reply to IceFloe [13] January 23, 2024 9:33 am

C++ only requires a `main()` function. It does not require that `main()` live in a file called `main.cpp`. However, putting `main()` inside `main.cpp` is conventional because it makes it easy to find.

Your compiler can compile .cpp files in any order it wants. It produces one object file per source file. There can be multiple main.cpp files as long as they are in different directories.

You can't view the source of an already compiled file. Save your source somewhere if you're going to want it later.

5 Reply

Turtleluver3000
January 19, 2024 6:19 pm

love the content

2 Reply

EMEN
January 11, 2024 12:27 pm

I love how every chapter isn't too long and has key concepts and then the next main term is for the next chapter. Makes learning not feel overburdening or too hectic all at once. Just enough reading and taking in of material in one go. The best site for learning C hands down :)

3 Reply

suprabobby
January 5, 2024 3:33 pm

loving this so far!

0 Reply

David
December 30, 2023 8:58 am

> "A library file is a collection of precompiled code that has been “packaged up” for reuse in other programs."

Does this mean that they are stored as object files?

3 Reply

Alex Author
Reply to David [14] January 2, 2024 12:17 pm

No. How library files are stored is platform specific.

1 Reply

Abhey
Reply to Alex [15] January 8, 2024 6:18 am

> but as it is stated to be 'precompiled' doesn't it simply mean it has been compiled beforehand and is consequently, simply stored as an object file
>
> 1 Reply

Alex Author

Reply to Abhey [16] January 8, 2024 4:03 pm

The precompiling of C++ source code does result in object files, but the "packaging up" of those files converts them to other platform-specific formats (executables, library files, etc...)

2 Reply

Led Schmidt

December 18, 2023 10:37 am

Is there a place in this or other sections of the lesson to specifically download ALL the necessary programs to begin "writing programs" and then all the things needing downloaded and installed to compete a program?. Also, with all due respect, the video where the gent is explaining downloading something, and then installing it is so hard to understand as to be useless.

0 Reply

Alex Author

Reply to Led Schmidt [17] December 19, 2023 8:50 pm

Yes, keep reading.

1 Reply

Oliver

December 17, 2023 7:46 pm

40 pushups B)

1 Reply

AbeerOrTwo

December 13, 2023 6:37 pm

Steady progress

1 Reply

Jeromey
December 1, 2023 2:37 am

New learner here! This is interesting!

1 Reply

Pratyaksh
November 22, 2023 4:58 am

I am new here to revise my concepts, I had learn C++ in my school and it's been 3 years now. After looking at comments I can't believe that many people are learning C++ right now wow !

1 Reply

Krishnakumar
October 2, 2023 5:05 am

In 2023, I think we ought to be referring users to something like build2 (or the current industry standard CMake) instead of Makefiles. Makefiles don't work natively on Windows, which is the OS used by most learners on this site (as per an earlier comment from the site's author).

Last edited 6 months ago by Krishnakumar

1 Reply

Alex Author
Reply to Krishnakumar [18] October 2, 2023 2:00 pm

Revised the comment to focus on build automation tools rather than Make specifically. Thanks!

2 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/introduction-to-cpp-development/
3. https://ikeamenu.com/humix/video/Ipms8sWsckf

4. https://www.learncpp.com/cpp-tutorial/installing-an-integrated-development-environment-ide/
5. https://www.learncpp.com/
6. https://www.learncpp.com/introduction-to-the-compiler-linker-and-libraries/
7. https://www.learncpp.com/cpp-tutorial/introduction-to-c17/
8. https://www.learncpp.com/cpp-tutorial/configuring-your-compiler-compiler-extensions/
9. https://gravatar.com/
10. https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/#comment-594692
11. https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/#comment-593726
12. https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/#comment-592711
13. https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/#comment-592666
14. https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/#comment-591522
15. https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/#comment-591651
16. https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/#comment-591954
17. https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/#comment-591148
18. https://www.learncpp.com/cpp-tutorial/introduction-to-the-compiler-linker-and-libraries/#comment-588048